UNIT-I — Full Stack Basics, JavaScript & Node.js Introduction

1. **Explain the Web Development Framework with neat architecture (User, Browser, Web Server, Backend Services).**
2. **Describe the Full Stack Components (Node.js, Express, Angular, MongoDB) and their roles.**
3. **Explain the Event-Driven Architecture of Node.js. How is it different from traditional multithreaded servers?**
4. **Discuss the Node.js Event Loop with a clear diagram and its working process.**
5. **Explain Node Package Manager (NPM). Describe installing, upgrading and using packages.**
6. **What are Callbacks in Node.js? Explain how asynchronous programming works using callbacks and event queue.**

UNIT-II — Node.js Concepts, Modules, HTTP & File System

1. **Explain JSON handling in Node.js** with examples of parsing and stringification.
2. **Explain the Buffer and Stream modules in Node.js**. Compare streams vs buffers.
3. **Describe File System Operations in Node.js**, including reading, writing, updating and deleting files.
4. **Explain the steps in creating HTTP Web Server in Node.js** using http module with example code.
5. **Discuss Query String and URL processing** in Node.js with examples.
6. **Write short notes on any two Node.js core modules:** (a) OS Module (b) DNS Module (c) Crypto Module.

UNIT-III — MongoDB & NoSQL

1. **Explain the need of NoSQL databases**. Compare SQL vs NoSQL with examples.
2. **Describe MongoDB data types and JSON document structure** with examples.
3. **Explain CRUD operations (Insert, Update, Delete, Find)** in MongoDB with suitable commands.
4. **Explain MongoDB Data Modeling**, including embedded documents vs references.
5. **Explain how to connect MongoDB with Node.js** using MongoClient.
6. **Discuss MongoDB User Management**, authentication and Access Control (roles & privileges).

UNIT-IV — Express & Angular

1. **Explain Express Framework**. How routing is implemented in Express with examples?
2. **Explain Request and Response objects** in Express.
3. **Describe the architecture of Angular** and its main components (Modules, Components, Services).
4. **Explain Data Binding in Angular** (interpolation, property, event, two-way).
5. **Explain Angular Directives** (Built-in & Custom) with examples.
6. **Discuss Angular Components** – lifecycle, decorators, template and style binding.

UNIT-V — React

1. **Explain the Features and Advantages of React**. Why is Virtual DOM needed?
2. **Describe React Component Types** (Class Components & Functional Components) with examples.
3. **Explain State and Props in React** with suitable examples.
4. **Discuss React Component Lifecycle Methods** with neat diagrams.
5. **Explain React Routing**. How to create single-page navigation using React Router?
6. **Explain Forms handling in React** and how to manage form input using controlled components.

UNIT-1

1 Q). Explain the Web Development Framework with Neat Architecture (User, Browser, Web Server, Backend Services).

Web development is a systematic process used to design, develop and deploy websites and web applications that can be accessed over the Internet. A Web Development Framework provides a structured way of building these applications by defining the flow between the **User**, **Browser**, **Web Server**, and **Backend Services (Database / APIs)**.
This architecture ensures smooth communication, high performance, security, and scalability.

In modern software engineering, understanding this architecture is essential because almost every full-stack application uses this model. Technologies like **HTML, CSS, JavaScript, Node.js, Angular, React, MongoDB, Express**, etc., work on the same architectural design.

1. INTRODUCTION TO WEB DEVELOPMENT FRAMEWORK

A Web Development Framework acts as the backbone of any web application. It defines:

- How the user interacts with the system
- How the browser sends requests
- How the server processes them
- How the backend (database, services) stores or retrieves information

It provides predefined functions, libraries, and tools so developers do not have to build everything from scratch.

Popular frameworks:

- **Frontend:** Angular, React, Vue.js
- **Backend:** Node.js with Express, Django (Python), Spring Boot (Java)
- **Database side:** MongoDB, MySQL, PostgreSQL

2. OVERALL ARCHITECTURE OF WEB DEVELOPMENT FRAMEWORK

A standard web application architecture contains four important components:
✔ **1. User (Client / End User)**
✔ **2. Browser (Frontend Interface)**
✔ **3. Web Server (Application Server)**
✔ **4. Backend Services (Database, APIs, Authentication Services)**
All these parts work together to deliver a complete and interactive web experience.

3. NEAT DIAGRAM OF WEB APPLICATION ARCHITECTURE

```
+---------------------+
|      USER       |
| (Human Visitor)  |
+---------+----------+
          |
          v
+------------------+
|     BROWSER      |
| (Chrome, Edge)   |
+---------+--------+
          |
  HTTP / HTTPS Request
          |
          v
+------------------+
|   WEB SERVER     |
| (Node.js/Express)|
+---------+--------+
          |
 Processes Logic, Routing
          |
          v
+---------------------+
|   BACKEND SERVICES   |
| DB, Authentication,  |
| File Storage, APIs   |
+---------------------+
          |
  Sends Data Back
          |
          v
+------------------+
|     BROWSER      |
+------------------+
          |
 Output shown to User
```

This layered architecture ensures **security, reusability, and scalability** of the application.

4. USER (CLIENT / END USER)

The user is the person who interacts with the web application.
Examples: a student checking results, a customer shopping online, an employee filling an attendance form.
Characteristics:

- Uses devices like mobile, laptop, tablet, PC
- Does not see the internal logic of the system
- Only interacts with the graphical interface

User interacts only with the **Frontend (Browser UI)**.

5. BROWSER (FRONTEND INTERFACE)

A Browser such as **Chrome, Firefox, Microsoft Edge** is a software that displays the front-end content of the website.

It performs three major tasks:

✔ 1. Rendering the Web Page (HTML, CSS)

The browser converts web code into a visible user interface.

✔ 2. Executing Client-Side Scripts (JavaScript)

JavaScript runs inside the browser to handle:

- Form validation
- Dynamic changes
- Button clicks
- Animations
- API calls (AJAX / Fetch API)

✔ 3. Sending Requests to Web Server (HTTP / HTTPS)

Whenever user clicks:

- Login
- View product
- Submit form

the browser sends a **request** to the server.

Example:

If a student clicks "Check Results", the browser sends a request:

```
GET /results?rollno=12345
```

6. WEB SERVER (APPLICATION SERVER)

A Web Server is a software/hardware which receives requests from the browser, processes them, and sends the output back.

Examples:

- **Node.js + Express Server** (in modern full stack applications)
- Apache
- Nginx
- IIS (Microsoft)

Main Responsibilities of Web Server

✔ 1. Routing

It decides which function to execute based on URL.

Example in Node.js Express:

```
app.get("/student", function(req, res){
    res.send("Student Information");
});
```

✔ 2. Processing Business Logic

The server contains the application rules, such as:

- Checking login credentials
- Calculating bill amounts
- Saving data to database
- Retrieving information

✔ 3. Security and Authentication

Ensures only authorized users can access sensitive data.

✔ 4. Communication with Backend Services

It contacts the database and other services to fetch data.

Example: Validating bank OTP, checking stock details, sending email confirmation.

7. BACKEND SERVICES (DATABASE & SUPPORT SERVICES)

Backend Services are the heart of the application where all data storage, retrieval, and heavy processing takes place.

Common Backend Components

✔ 1. Database

Stores permanent data.

- MongoDB (NoSQL)
- MySQL (SQL)
- PostgreSQL
- Oracle DB

Example stored data:

- Student details
- Online orders
- Product information
- Login credentials (encrypted)

✔ 2. Authentication Service

Used for login, OTP, user roles etc.

Examples:

- JWT tokens
- OAuth
- Firebase Auth

✔ 3. File Storage

Used for storing images, documents, etc.

✔ 4. External APIs

Used for:

- Weather information
- Payment gateway (Razorpay, Paytm)
- SMS services (Twilio)
- Google Maps API

8. COMPLETE FLOW OF WEB APPLICATION

Let us understand the flow with a real-life Indian example:

Example: Student Portal Login

Step 1: User Action

A student opens JNTUH Results Portal and clicks **"Login"**.

Step 2: Browser Sends Request

Browser sends:

POST /login with username & password.

Step 3: Web Server Receives Request

Node.js server checks:

- URL
- Method
- Data

Then sends the data to backend services.

Step 4: Backend Processing

Database (MongoDB/MySQL) validates:

- Whether the student exists
- Whether password matches

Step 5: Server Generates Response

If login successful:

"Login Successful. Welcome Suresh!"

If invalid:
"Incorrect Password".

Step 6: Browser Displays Output

User sees the result on the screen.

9. ADVANTAGES OF WEB DEVELOPMENT ARCHITECTURE

✔ 1. High Scalability
Thousands of users can use simultaneously.

✔ 2. Security
Separation of UI, server, and database protects sensitive data.

✔ 3. Flexibility
Frontend and backend can be updated independently.

✔ 4. Faster Performance
Server handles processing; browser handles display.

✔ 5. Code Reusability
Frameworks provide reusable modules.

10. CONCLUSION

The Web Development Framework architecture is the backbone of all modern web applications.
It clearly defines how the **User**, **Browser**, **Web Server**, and **Backend Services** interact with each other.
This structure ensures smooth communication, secure data handling, consistent performance, and easy development.

Understanding this architecture is essential for Full Stack Developers, especially in technologies like **Node.js, Express, Angular, React, and MongoDB**, which follow the same layered model.

2Q.) Describe the Full Stack Components (Node.js, Express, Angular, MongoDB) and their roles.

Full Stack Development refers to the development of both **client-side (frontend)** and **server-side (backend)** parts of a web application. A full stack developer works on all layers of the application, including **UI, server logic, database, and integration**.

Modern web applications mostly follow the **MEAN** or **MERN** stack. In JNTUH syllabus, the commonly used Full Stack Components are:

1. **Node.js – Backend Runtime Environment**
2. **Express – Backend Framework**
3. **Angular – Frontend Framework**
4. **MongoDB – NoSQL Database**

These four technologies together make a powerful combination for designing scalable, efficient, and interactive applications.

1. INTRODUCTION TO FULL STACK COMPONENTS

Full stack components help in developing:

- Single Page Applications (SPAs)
- Real-Time Applications
- Highly scalable cloud-based apps
- E-commerce and social media platforms
- Mobile-friendly responsive web apps

Each component plays a unique role in complete application development.

2. ARCHITECTURE OF FULL STACK DEVELOPMENT

```
+--------------------+
|      Angular       |
|  (Frontend UI)     |
+----------+---------+
           |
           | HTTP Requests (GET/POST)
           v
+--------------------+
|      Express       |
| (Server Framework) |
+----------+---------+
           |
           | Node.js Runtime
           v
+--------------------+
|      Node.js       |
| (Backend Logic)    |
+----------+---------+
           |
           | DB Driver
           v
+--------------------+
|      MongoDB       |
|  (NoSQL Database)  |
+--------------------+
```

This architecture clearly separates each layer while allowing smooth communication.

3. NODE.JS – BACKEND RUNTIME ENVIRONMENT

Node.js is an **open-source, cross-platform JavaScript runtime environment** built on Google Chrome's V8 engine.
It allows developers to run JavaScript on the server side.

✔ Key Features of Node.js

1. **Asynchronous and Event Driven**
   Node.js uses non-blocking I/O, meaning multiple users can be handled simultaneously.
2. **Fast Execution**
   V8 engine compiles JavaScript directly to machine code.
3. **NPM (Node Package Manager)**
   Provides thousands of reusable libraries like express, mongoose, bcrypt, etc.
4. **Single Threaded but Highly Scalable**
   Uses an event loop to manage requests efficiently.

Role of Node.js in Full Stack Development

- Executes backend business logic
- Handles user requests (login, registration, search, etc.)
- Performs file operations
- Manages APIs for AJAX and fetch calls

- Handles authentication, authorization
- Communicates with databases
- Generates dynamic responses for the frontend

Example Code (Node.js Server):

```
const http = require('http');
http.createServer((req, res) => {
  res.end("Server Running Successfully!");
}).listen(3000);
```

4. EXPRESS – NODE.JS WEB APPLICATION FRAMEWORK

Express.js is a lightweight, minimal, and flexible framework built on Node.js.

✔ Features of Express

1. **Routing System** for handling URLs
2. **Middleware Support** for functions like logging, authentication, etc.
3. **Template Engines** support for dynamic HTML
4. **Easy Integration** with MongoDB and Angular
5. **Simplifies Server Code**

Role of Express in Full Stack Development

Express acts as the **middle layer** between frontend and backend.

- Handles HTTP requests (GET, POST, PUT, DELETE)
- Defines routes for each operation
- Processes form data and JSON data
- Validates inputs
- Calls database functions
- Sends output back to Angular frontend

Example of Routing in Express:

```
app.get("/students", function(req, res){
  res.send("List of Students");
});
```

5. ANGULAR – FRONTEND FRAMEWORK

Angular is a **TypeScript-based front-end framework** maintained by Google.
It is used for building **Single Page Applications (SPA)**.

✔ Features of Angular

1. **Components-Based Architecture**
2. **Two-Way Data Binding**
3. **Directives** for dynamic HTML
4. **Services and Dependency Injection**
5. **Routing for SPA Navigation**
6. **Pipes** for data transformation

Role of Angular in Full Stack Development

- Displays the user interface
- Handles user interactions
- Sends API requests to Express
- Shows dynamic content without page reload
- Validates user input
- Provides SPA features for faster performance

Example Angular Component Code:

```
export class AppComponent {
  title = "Welcome to Angular App!";
}
```

Practical Example

If user clicks **“Get Students List”**, Angular sends request:

GET http://localhost:3000/students

Server fetches data from MongoDB and returns it, which Angular displays instantly.

6. MONGODB – NOSQL DATABASE

MongoDB is a document-oriented **NoSQL database** that stores data in **JSON-like BSON format**.

✔ Features of MongoDB

1. **Schema-less Structure** – Flexible documents
2. **High Speed & Scalability**
3. **Collections instead of Tables**
4. **JSON Document Storage**
5. **Automatic Sharding and Replication**

Role of MongoDB in Full Stack Development

- Stores user information
- Stores dynamic application data (products, logs, messages)
- Handles large unstructured data efficiently
- Communicates easily with Node.js through drivers
- Allows fast read/write operations

Example MongoDB Document:

```
{
  "name": "Rohit",
  "branch": "CSE",
  "year": 3
}
```

7. COMBINED ROLE OF ALL COMPONENTS (MEAN Stack Flow)

✔ Step 1: User interacts with Angular UI

Clicks Login → Angular collects data
✔ **Step 2: Angular sends HTTP request**
POST /login to Express server
✔ **Step 3: Express processes the request**
Validates input → Calls Node.js controller
✔ **Step 4: Node.js interacts with MongoDB**
Queries database to authenticate user
✔ **Step 5: MongoDB returns data**
Sends result back to Node.js
✔ **Step 6: Server sends response**
Express → Angular → User sees output

8. REAL-LIFE EXAMPLE – ONLINE SHOPPING APP

Scenario: User orders a mobile phone

Component	**Role**
Angular	Displays product list, shows cart, collects order details
Express	Receives order request, validates data
Node.js	Applies business logic, processes order
MongoDB	Stores order details, user info, payment status

This division of responsibility makes the application **scalable, fast, and maintainable**.

9. CONCLUSION

Node.js, Express, Angular, and MongoDB form the core of modern full-stack development. Their roles are clearly defined:

- **Angular** handles UI and user interactions
- **Express** manages routing and HTTP handling
- **Node.js** works as the backend logic engine
- **MongoDB** stores and processes data

Together, they provide a complete environment for developing efficient, interactive, secure, and scalable web applications.

This combination is widely used in engineering projects, real-time applications, and industry-based systems, making it a vital part of full stack development.

3Q). Explain the Event-Driven Architecture of Node.js. How is it different from traditional multithreaded servers?

Answer:

Node.js is widely used for developing fast, scalable, and real-time web applications.
The main reason behind its high performance is its **Event-Driven Architecture**, which uses a **single-threaded, non-blocking I/O model**.

Instead of creating separate threads for each request (like traditional servers), Node.js handles all requests using an **event loop** and **callbacks**. This makes it extremely efficient for network applications.

1. INTRODUCTION TO EVENT-DRIVEN ARCHITECTURE

Event-driven architecture means:

- Every operation in Node.js is treated as an **event**
- When an event occurs, Node.js triggers a **callback function**
- Non-blocking I/O ensures the main thread never waits

This model is used in:

- Chat applications
- Online gaming
- Streaming platforms
- Realtime dashboards
- Microservices
- IoT applications

2. ARCHITECTURE OF NODE.JS (EVENT-DRIVEN MODEL)

Node.js uses four main components:

1. **Event Loop**
2. **Event Queue**
3. **Callbacks**
4. **Worker Threads (for heavy tasks)**

Below is the simplified architecture.

3. NEAT DIAGRAM OF EVENT-DRIVEN ARCHITECTURE

```
          +-------------------------+
          |     Node.js Process     |
          +-----------+-------------+
                      |
               Incoming Requests
                      |
                      v
          +-------------------------+
          |     Event Queue         |
          +-----------+-------------+
                      |
                      v
          +-------------------------+
          |       Event Loop        |
          +-----------+-------------+
                      |
    ---------------------------------------------
    |             |                           |
    v             v                           v
 Immediate     Non-blocking I/O          Timer Events
 Callbacks      (File/DB/Network)        setTimeout()
                  |
                  v
          +------------------------+
          |   Worker Pool (C++)    |
          +------------------------+
                      |
               Returns Result
                      |
                      v
             Callback Executed
```

4. WORKING PRINCIPLE (STEP-BY-STEP)

Step 1: Client sends a request
Example: Fetching student details
GET /student?id=104
Step 2: Request enters the Event Queue
Node.js pushes this request into the **Event Queue**.
Step 3: Event Loop picks the request
The Event Loop continuously checks:

- Are there any pending callbacks?
- Are there any asynchronous tasks pending?

Step 4: If request is non-blocking → handled immediately
Example: Routing, logging, sending a response, etc.
Step 5: If request requires I/O → Sent to Worker Pool
Examples:

- Reading a file
- Accessing a database
- Network calls

A separate worker thread handles the operation.
Step 6: Result returned → Callback executed
Once done, Node.js executes the callback and sends the response back to the user.

5. SIMPLE EXAMPLE OF EVENT-DRIVEN CODE

```
const fs = require('fs');

console.log("Start");

fs.readFile("data.txt", function(err, data){
   console.log("File read complete");
});

console.log("End");
```

Output:

```
Start
End
File read complete
```

Explanation:

- File reading goes to Worker Pool
- Main thread continues without blocking
- Callback is executed when file reading completes

This demonstrates **non-blocking behavior**.

6. ADVANTAGES OF EVENT-DRIVEN ARCHITECTURE

✔ 1. High Scalability
A single thread handles thousands of requests.
✔ 2. Faster Response Time
Event loop prevents waiting.
✔ 3. Memory Efficient
No need to create multiple threads.
✔ 4. Highly Suitable for I/O Applications
Because most operations are non-blocking.
✔ 5. Simplifies Server Development
Callbacks, events, and promises make code clean.

7. TRADITIONAL MULTITHREADED SERVERS (E.g., Java, PHP, .NET)
Traditional servers follow a different model:
✔ Each request gets its own thread
This thread:

- Handles logic
- Performs I/O
- Waits for DB responses
- Uses memory

✔ Thread Pool overhead
If 10,000 users connect:

- 10,000 threads may be created
- High CPU and RAM usage
- Context switching overhead

✔ Blocking I/O Model
Server *waits* for operations like:

- File read
- Database query
- API response

This reduces performance in high-load environments.

8. DIFFERENCE BETWEEN NODE.JS AND MULTITHREADED SERVERS

Feature	Node.js (Event-Driven)	Traditional Servers (Multithreaded)
Thread Model	Single-threaded	Multi-threaded
I/O Model	Non-blocking	Blocking
Request Handling	Event loop + callbacks	Each request gets separate thread
Scalability	Very high	Medium
Memory Usage	Low	High
Performance	Excellent for I/O	Good for CPU tasks
Concurrency	Event-based	Thread-based
Context Switching	No	Yes (CPU overhead)
Examples	Node.js, Nginx	Apache, Tomcat, PHP-FPM

9. REAL-LIFE EXAMPLE COMPARISON

Case: 10,000 users requesting data simultaneously

✔ Traditional Server

- 10,000 threads → high RAM usage
- CPU wastes time switching threads
- Server becomes slow

✔ Node.js

- Single thread handles all
- No context switching
- Event loop dispatches work efficiently
- Smooth performance

This is why companies like **Netflix, PayPal, LinkedIn** use Node.js for handling millions of requests.

10. WHY NODE.JS IS BETTER FOR MODERN APPLICATIONS

Node.js is well-suited for:

- Chat applications
- Real-time messaging
- Live streaming
- Social media feeds
- Online banking dashboards
- IoT devices

Because they need fast updates without waiting for threads.

11. CONCLUSION

Node.js uses a powerful **event-driven, non-blocking architecture**, enabling it to handle thousands of concurrent requests using a single thread. The Event Loop, Event Queue, Callbacks, and Worker Pool work together to ensure maximum performance.

In contrast, traditional multithreaded servers allocate one thread per request, causing heavy memory usage and reduced scalability.

Thus, Node.js is ideal for modern I/O-intensive, real-time applications, whereas multithreaded servers are more suitable for CPU-intensive tasks.

4Q.) Discuss the Node.js Event Loop with a clear diagram and its working process.

Node.js is a powerful backend platform built on the Chrome V8 JavaScript engine. One of the main reasons for its high performance and scalability is the **Event Loop**, which enables Node.js to handle multiple concurrent requests using a **single thread**.

The Event Loop is the "heart" of Node.js' asynchronous architecture. It continuously monitors tasks, manages callbacks, executes events, and delegates time-consuming operations to background workers.

1. INTRODUCTION TO EVENT LOOP

The **Event Loop** is a mechanism that allows Node.js to perform **non-blocking, asynchronous operations**, even though JavaScript runs in a **single thread**.

Instead of creating multiple threads, Node.js uses:

- **Event Loop**
- **Callback Queue**
- **Timers**
- **Worker Pool (libuv thread pool)**

This architecture allows thousands of clients to be served efficiently.

2. NEED FOR EVENT LOOP

The event loop solves the biggest problem in traditional synchronous programming:

"If one request is slow, the entire system should not stop."
Node.js achieves this by:

- Offloading I/O tasks
- Executing callbacks only when results are ready
- Allowing the main thread to keep processing new incoming requests

This makes Node.js ideal for real-time applications like chat, streaming, and gaming.

3. NEAT DIAGRAM OF NODE.JS EVENT LOOP

```
          +-------------------------------------+
          |          Node.js Process            |
          +-------------------------------------+
                      |
               Incoming Requests
                      |
                      v
          +-------------------------------------+
          |            Event Queue              |
          +------------------+------------------+
                             |
                             v
          +-------------------------------------+
          |            Event Loop               |
          +------------------+------------------+
                             |
    ---------------------------------------------------------------------
    |                 |                     |                        |
    v                 v                     v                        v
  Timers        Pending Callbacks      I/O Polling              Check Phase
(setTimeout,     Completed callbacks    Network/File I/O        setImmediate()
 setInterval)      waiting              operations
                              |
                              |
                              v
          +-------------------------------------+
          |      Worker Pool (libuv C++)        |
          | (File I/O, DNS, heavy operations)   |
          +-------------------------------------+
                             |
                       Returns result
                             |
                             v
          +-------------------------------------+
          |      Callback Queue                 |
          +-------------------------------------+
                             |
                             v
                     Executed by Event Loop
```

4. PHASES OF THE EVENT LOOP

The event loop runs in **multiple phases**, each handling specific types of callbacks.

Below is the complete flow.

Phase 1: Timers Phase

Handles callbacks from:

- setTimeout()
- setInterval()

Example:

```
setTimeout(() => {
  console.log("Timer executed");
}, 1000);
```

These callbacks run in this phase when their time expires.

Phase 2: Pending Callbacks

Handles I/O-related callbacks that are deferred by system operations.

Example: TCP errors, DNS lookup callbacks, etc.

Phase 3: Idle, Prepare Phase

Internal use by Node.js; not accessible to developers.

Phase 4: Poll Phase (MOST IMPORTANT PHASE)

This is where Node.js spends most of its time.

Responsibilities:

- Retrieve new I/O events
- Execute I/O callbacks
- When no events, Node.js waits here

Example I/O operations handled:

- Reading files
- Database queries
- Network requests

Phase 5: Check Phase

Executes callbacks scheduled by **setImmediate()**.

Example:

```
setImmediate(() => {
  console.log("Immediate callback");
});
```

Phase 6: Close Callbacks Phase

Handles:

- socket.on('close', ...)
- db.close() callbacks
- WebSocket closure events

---

5. ROLE OF WORKER POOL (libuv)

Although Node.js is single-threaded, heavy operations are handled by the **libuv thread pool**.

Used for:

- File system operations
- DNS operations
- Compression
- Encryption (crypto module)

These run on background threads and return results to the Event Loop.

---

6. COMPLETE WORKING PROCESS OF EVENT LOOP (STEP BY STEP)

Let us consider an example program:

```
console.log("Start");

setTimeout(() => {
   console.log("Timer Complete");
}, 0);

fs.readFile("data.txt", () => {
   console.log("File Read Complete");
});

console.log("End");
```

Step-by-Step Execution

Step 1: "Start" printed

Runs immediately (synchronous).

Step 2: setTimeout() callback → Timers Phase

Even with 0 ms delay, the callback is placed in the Timers Queue.

Step 3: fs.readFile() → Worker Pool

File reading goes to background thread.

Step 4: "End" printed

Main thread continues without waiting.

Step 5: Poll Phase

Waits until file reading completes.

Step 6: File read complete → Callback queued

Callback enters Pending I/O Queue.

Step 7: Timer callback executed

Because setTimeout(0) executes after the Poll Phase.

Step 8: File read callback executed

Final Output:

```
Start
End
Timer Complete
File Read Complete
```

---

7. ADVANTAGES OF EVENT LOOP

✔ Non-blocking Architecture

Multiple tasks processed concurrently.

✔ Highly Scalable

One thread handles thousands of users.

✔ Efficient Memory Usage

No cost of creating threads.

✔ Best for I/O Applications

Reading files, DB queries, API calls.

---

8. LIMITATIONS OF EVENT LOOP

✕Not ideal for CPU-intensive tasks

Because it blocks the single thread.

✕Requires careful handling of callbacks

Callback hell may occur (solved using Promises/Async-Await).

---

9. CONCLUSION

The Node.js Event Loop is the core engine that enables asynchronous, non-blocking operations in a single-threaded environment. By using phases, callbacks, worker pool, and an efficient event-driven structure, it ensures:

- High performance
- Scalability
- Smooth handling of I/O operations

This makes Node.js extremely powerful for real-time and large-scale network applications.

5Q). Explain Node Package Manager (NPM). Describe installing, upgrading and using packages.

Node Package Manager (NPM) is one of the most important tools used in the Node.js ecosystem. It is the official package manager for Node.js and acts as an online repository where thousands of reusable open-source libraries (called packages or modules) are available.

NPM helps developers to install, update, manage and publish JavaScript packages required for server-side as well as frontend development. It greatly simplifies the development process because programmers do not need to write code from scratch.

1. INTRODUCTION TO NPM

NPM stands for **Node Package Manager**. It has two major parts:

a) Online Repository (https://www.npmjs.com/)

This contains lakhs of packages developed by the Node.js community.

b) Command-Line Tool (npm command)

Used to install, update or uninstall packages from the project.

Every Node.js developer regularly works with NPM because it:

- Saves time
- Provides reliable third-party libraries
- Ensures good project structure
- Supports version control
- Allows code sharing

2. FEATURES OF NPM

✔ 1. Package Installation

Install any library in seconds.

✔ 2. Dependency Management

NPM automatically manages all required dependencies.

✔ 3. Version Control

Supports semantic versioning (major.minor.patch).

✔ 4. Publishing Packages

Developers can publish their own modules for others.

✔ 5. Scripts Execution

Allows creation of custom commands in *package.json*.

✔ 6. Global and Local Installation

Packages can be installed system-wide or project-specific.

3. NPM ARCHITECTURE DIAGRAM

```
+--------------------------+
|   NPM Registry Server    |
| (Online Repository)      |
+------------+-------------+
             |
      Download Packages
             |
             v
+--------------------------+
|      NPM CLI Tool        |
| (npm install, update)    |
+------------+-------------+
             |
             v
+--------------------------+
|   Project Node Modules   |
|      (node_modules/)     |
+--------------------------+
```

4. PACKAGE.JSON — HEART OF EVERY NPM PROJECT

When a Node.js project starts, we create a package.json file using:

```
npm init
```

This file stores:

- Project name
- Version
- List of dependencies

- Scripts
- Author name

Example package.json snippet:

```
{
  "name": "myapp",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

5. INSTALLING PACKAGES USING NPM

There are two types of installations:

A) Local Installation (Project-Specific)

The package is stored inside the project’s **node_modules** folder.

Command:

```
npm install <package-name>
```

Example:

```
npm install express
```

This adds express to:

- node_modules folder
- package.json → dependencies

Example Use:

```
const express = require("express");
const app = express();
```

B) Global Installation (System-Wide)

These packages can be used in any project.

Command:

```
npm install -g <package-name>
```

Example:

```
npm install -g nodemon
```

Nodemon can now be used globally to auto-restart the server.

C) Installing Specific Version

npm install express@4.17.0

Useful for compatibility with old applications.

D) Installing All Dependencies

If we download a project from GitHub:

npm install

It automatically reads package.json and installs all required packages.

6. UPGRADING PACKAGES USING NPM

Updating packages ensures better performance, bug fixes, and security.

A) Updating a Specific Package
npm update <package-name>

Example:

npm update mongoose

B) Updating All Packages
npm update

This updates all dependencies listed in package.json.

C) Checking Package Versions
npm outdated

This shows:

- Current version
- Wanted version
- Latest version

D) Updating Global Packages
npm update -g <package-name>

Example:

npm update -g nodemon

7. USING PACKAGES IN A NODE.JS PROGRAM

After installation, packages can be used by importing them using require() or ES modules.

Example 1: Using Express

```
const express = require("express");
const app = express();

app.get("/", (req, res) => {
   res.send("Welcome to Express!");
});

app.listen(3000, () => console.log("Server running"));
```

Example 2: Using Axios package

Install:
```
npm install axios
```
Use:
```
const axios = require("axios");

axios.get("https://api.github.com/users")
     .then(response => console.log(response.data));
```

Example 3: Using Nodemon (Globally Installed)

Start server with auto-reload:

```
nodemon server.js
```

8. UNINSTALLING PACKAGES

Local Uninstall:
```
npm uninstall <package-name>
```

Global Uninstall:

```
npm uninstall -g <package-name>
```

9. ADVANTAGES OF USING NPM

✔ **Saves development time**
✔ **Provides high-quality libraries**
✔ **Automatic dependency installation**
✔ **Easy version management**
✔ **Huge community support**
✔ **Helps build large-scale full stack applications easily**

CONCLUSION

NPM is a powerful tool that simplifies Node.js development. It manages project dependencies, installs reusable packages, upgrades them, and provides a smooth workflow for developers. Whether for backend frameworks like Express or frontend tools like Angular CLI and React, NPM plays a crucial role in modern full stack development.

It is essential for every engineering student and full stack developer to understand how to install, update, and manage NPM packages effectively.

6Q). What are Callbacks in Node.js? Explain how asynchronous programming works using callbacks and event queue.

Answer:

Node.js is well-known for its high performance and scalability. The main reason for this performance is its **asynchronous, non-blocking architecture**, which is implemented using **callbacks** and the **event queue**. Callbacks are one of the fundamental concepts in Node.js and form the base of its event-driven programming model.

1. WHAT ARE CALLBACKS IN NODE.JS?

A **callback** is a function passed as an argument to another function.
It will be executed **after** the completion of an operation, especially for asynchronous tasks.

Definition (exam-friendly):

A callback in Node.js is a function that gets executed only after the completion of a non-blocking asynchronous operation, such as reading files, querying a database, network calls, timers, etc.

Node.js uses callbacks to avoid blocking the main thread, so the application does not wait for any slow operation.

2. SIMPLE EXAMPLE OF CALLBACK IN NODE.JS

```
function greet(name, callback){
   console.log("Hello " + name);
   callback();
}

greet("Suresh", function(){
   console.log("Welcome to Node.js");
});
```

Output:

```
Hello Suresh
Welcome to Node.js
```

Here, the second function is executed only after the first function finishes.

3. NEED FOR CALLBACKS

Callbacks are required because:

- Node.js is **single-threaded**
- I/O operations like file reading and database queries take time
- Blocking the thread will block all other users

Callbacks allow Node.js to perform heavy tasks in the background while the main thread continues running other tasks.

4. ASYNCHRONOUS PROGRAMMING IN NODE.JS

Node.js runs JavaScript in a **non-blocking** manner.
This means tasks are executed without stopping the main program.

Synchronous Example (Blocking)

```
let data = fs.readFileSync("file.txt");
console.log(data);
console.log("After File Read");
```

Here, the second console output waits until file reading completes.

Asynchronous Example (Non-Blocking)

```
fs.readFile("file.txt", function(err, data){
   console.log("File Read Complete");
});
console.log("After File Read");
```

Output:

```
After File Read
File Read Complete
```

The program **does not wait**.
The file is read in the background, and the callback runs after completion.

5. ROLE OF EVENT QUEUE AND EVENT LOOP

Node.js does not run asynchronous operations itself.
It uses the following components:

1. **Call Stack**
2. **Event Queue**
3. **Event Loop**
4. **Worker Pool (libuv)**

These work together to create asynchronous programming.

6. DIAGRAM OF ASYNCHRONOUS CALL USING CALLBACK

```
+------------------------+
|      Node.js App       |
+-----------+------------+
            |
   Call Function (fs.readFile)
            |
            v
+------------------------+
|     Worker Pool        |
| (File I/O, DB, DNS)    |
+-----------+------------+
            |
   Operation Completed
            |
            v
+------------------------+
|      Event Queue       |
+-----------+------------+
          |
    Event Loop Checks
          |
          v
+------------------------+
|    Execute Callback    |
+------------------------+
```

7. HOW ASYNCHRONOUS PROGRAMMING WORKS INTERNALLY (STEP BY STEP)

Let us understand step-by-step what happens when Node.js executes an asynchronous function with a callback.

Step 1: Function Call

```
fs.readFile("test.txt", callback);
```

This function is called from JavaScript code.

Step 2: I/O Operation Handed to Worker Pool

Node.js does **not** read files itself.
It passes the task to the **libuv Worker Pool**.
Main thread is now **free to continue**.

Step 3: Once I/O Completes → Callback Sent to Event Queue

After reading the file, the worker returns the result.
Callback is now pushed into the **Event Queue**.

Step 4: Event Loop Monitors the Queue

Event Loop continuously checks if:

- Call stack is empty
- Any callbacks are waiting

Step 5: Event Loop Executes Callback

Once the main thread is free, the callback function is executed.
This ensures Node.js handles many requests efficiently.

8. EXAMPLE SHOWING ASYNCHRONOUS CALLBACK BEHAVIOR

```
console.log("Start");

fs.readFile("data.txt", function(err, data){
   console.log("File reading completed");
});

console.log("End");
```

Output:

```
Start
End
File reading completed
```

Although file reading is written second, its callback runs last because:

- It is asynchronous
- It goes to worker pool
- Callback waits in Event Queue
- Event Loop executes it later

9. ADVANTAGES OF CALLBACKS AND EVENT QUEUE

✔ 1. High Performance
Allows handling thousands of users simultaneously.
✔ 2. Non-Blocking Architecture
Slow tasks do not block the main thread.
✔ 3. Efficient Resource Usage
Only one thread handles multiple requests.
✔ 4. Best for I/O Intensive Applications
Database, API calls, file reading, etc.

10. DISADVANTAGES (Callback Hell)

Nested callbacks may lead to:

- Difficulty in reading code
- Hard-to-fix bugs
- Pyramid-shaped structure

Example of Callback Hell:

```
task1(function(){
    task2(function(){
        task3(function(){
            task4();
        });
    });
});
```

This is solved using:

- **Promises**
- **Async/Await**

CONCLUSION

Callbacks play a crucial role in Node.js asynchronous architecture. They allow the program to continue executing without waiting for time-consuming operations. The Event Queue, Event Loop, and Worker Pool work together to process asynchronous tasks efficiently.

This model makes Node.js highly scalable, fast, and ideal for real-time applications like chats, dashboards, streaming services, and cloud platforms.

UNIT – I: Short Questions & Answers

1. What is Full Stack Development?

Full Stack Development refers to developing both frontend (client-side) and backend (server-side) of a web application. A full stack developer handles UI, server logic, database, and APIs.

2. What are Full Stack Components?

Full stack components include frontend frameworks like Angular/React, backend technologies like Node.js/Express, and databases like MongoDB. Together they form a complete application.

3. What is a Web Server?

A web server receives client requests, processes backend logic, interacts with the database, and sends responses. Examples: Node.js, Apache, Nginx.

4. What is Browser Rendering?

Browser rendering means converting HTML, CSS, and JavaScript code into a visible webpage. Chrome, Edge, and Firefox use rendering engines for this.

5. What is Node.js?

Node.js is a JavaScript runtime environment used to run JavaScript on the server side. It uses the V8 engine and supports asynchronous programming.

6. What is Angular used for?

Angular is a frontend framework used to build Single Page Applications (SPA). It supports components, data binding, directives, and routing.

7. What is Express.js?

Express is a lightweight Node.js web framework used to create APIs, handle routing, and manage server-side logic easily.

8. What is MongoDB?

MongoDB is a NoSQL database that stores data in JSON-like documents. It is highly scalable and used in modern web applications.

9. What is Asynchronous Programming?

Asynchronous programming allows tasks to run in the background without blocking execution. Node.js uses callbacks, promises, and event loop for this.

10. What is an Event Loop?

The Event Loop handles asynchronous operations in Node.js. It continuously checks event queues and executes callbacks when tasks are completed.

11. What is a Callback Function?

A callback is a function passed as an argument to another function. It runs after an asynchronous operation completes, like file reading.

12. What is NPM?

NPM stands for Node Package Manager. It is used to install, update, and manage JavaScript libraries required for Node.js development.

13. What is a Package in NPM?

A package is a reusable piece of code or library available through NPM. Examples: express, mongoose, nodemon.

14. What is package.json?

package.json is a configuration file that stores project details, version, dependencies, and scripts for a Node.js project.

15. What is a Single Page Application (SPA)?

SPA loads only one HTML page and updates the content dynamically without refreshing. Angular and React are used to build SPAs.

16. What is JavaScript Event-Driven Programming?

It means JavaScript responds to events (clicks, requests, timers) using callbacks or listeners. Node.js uses this model for server-side tasks.

17. What is the role of Backend Services?

Backend services handle business logic, authentication, database queries, and API responses. They run on servers like Node.js.

18. What is an API?

API (Application Programming Interface) allows communication between frontend and backend. It sends and receives data using HTTP requests.

19. What is Routing in a Web Application?

Routing means defining different URLs and linking them to specific actions or functions. Express.js handles routing in Node.js.

20. What is the Node.js Event Model?

Node.js uses an event-driven model where tasks are executed based on events triggered. The event loop manages asynchronous callbacks.